Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

**IIC2343 – Arquitectura de Computadores**

# Ayudantía 1

**Profesor: Yadran Eterovic**

**Ayudantes: Daniel Leal (dlleal@uc.cl), Jessica Hormazabal (jyhormazabal@uc.cl)**

## Preguntas

1. **Representaciones numéricas**

   a. **(T1 - II/2018)** Indique la base $\beta$ en la cual la siguiente ecuación es correcta:

   $$7_\beta + 8_\beta = 13_\beta$$

   Para resolver este ejercicio, debemos darnos cuenta de que la ecuación se simplifica al expandirla expresando cada número en la base $\beta$.

   $$\begin{aligned} 7_\beta + 8_\beta &= 13_\beta \\ 7 \times \beta^0 + 8 \times \beta^0 &= 1 \times \beta^1 + 3 \times \beta^0 \\ 7 + 8 &= \beta + 3 \\ \beta + 3 &= 15 \\ \beta &= 12 \end{aligned}$$

   b. Escriba el valor binario de los números decimales 7, 10 y 2.

   Para el número 7, calculamos que

   $$1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7$$

   Por lo que el valor binario del decimal 7 es 111.

   Para el número 10, calculamos que

   $$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 10$$

   Por lo que el valor binario del decimal 10 es 1010.

Para el número 2, calculamos que

$$1 \times 2^1 + 0 \times 2^0 = 2$$

Por lo que el valor binario del decimal 2 es 10.

c. **(I1 - II/2014)** Describa el valor decimal del número hexadecimal 0x94A6, si este se interpreta como binario con signo.

Una forma rápida de expresar un número hexadecimal como uno binario, es transformando cada dígito de este a base binaria con 4 dígitos (recordando que $2^4 = 16$):

- $9 = 1001_2$
- $4 = 0100_2$
- $A = 1010_2$
- $6 = 0110_2$

Luego, nuestro número en base binaria corresponde a $1001010010100110_2$. Como este se interpreta como binario con signo, y el bit más significativo corresponde a un 1, utilizamos el complemento a 2 para obtener la representación correcta:

$$1001010010100110_2 = -C_2(1001010010100110_2) = -0110101101011010_2 = -27482$$

2. **Operaciones binarias**

a. **(I1 - II/2011)** Dados $A = 45$ y $B = 57$, ¿cuál es el resultado, en binario, de la operación $A - B$?

Notemos que si queremos representar estos números en binario, y estamos realizando una operación que implica una resta, entonces necesariamente debemos considerar el bit de signo. Tenemos entonces que $A = 0101101_2$ y $B = 0111001_2$. Luego, restarle a un número positivo uno negativo es equivalente a sumarle su complemento a 2. Entonces:

$$-B = C_2(0111001_2) = 1000111_2$$
$$A - B = 0101101_2 + 1000111_2 = 1110100_2$$

Ahora, este número es negativo, por lo que si queremos su valor decimal correspondiente, realizamos nuevamente el complemento a 2:

$$A - B = -C_2(1110100_2) = -0001100_2 = -12$$

Finalmente, vemos que el número obtenido en binario es consistente con el resultado de la operación en base decimal.

b. Suponga que se tiene un total de 6 bits, usados para representar números positivos y negativos. Dados $A = 27$ y $B = 8$, ¿cuál es el resultado, en binario, de la operación $A + B$? ¿Por qué da este resultado?

Si se tiene un total de 6 bits, podemos representar sin problemas los A y B dados como números positivos. Tenemos entonces que $A = 011011_2$ y $B = 000100_2$. Luego, al realizar la operación:

$$A + B = 011011_2 + 001000_2 = 100011_2$$

Podemos ver que el resultado, utilizando representación binaria con signo, es negativo. El número, en base decimal, sería entonces:

$$100011_2 = -C_2(100011_2) = -011101_2 = -29$$

Esto sucede debido a que sobrepasamos nuestro poder de representación. Con 6 bits, el máximo número que podemos representar es $011111_2 = 31$, Si la suma nos da un resultado mayor a ese, al no ser capaces de representar dicho número, obtenemos un número incorrecto (pues la suma de dos números positivos no pueden dar como resultado uno negativo). Este resultado se conoce como **overflow**.

3. **Compuertas lógicas**

a. **(Apuntes - Operaciones aritméticas y lógicas)** Implemente un circuito 2 bit Multiplier, que realice la multiplicación entre dos valores de 2 bits.

Para ver esto de forma sencilla, primero vemos cómo implementar una multiplicación entre dos números de un bit:

| $A$ | $B$ | $A * B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Es fácil ver que la multiplicación la podemos realizar a partir de la compuerta lógica `AND`. Ahora, como nos piden una multiplicación entre dos números de dos bits, tenemos que seguir el método de multiplicación tradicional. Para cumplir este objetivo es importante recordar la composición de los *half-adders*, en la que la suma entre dos números se representa por la compuerta `XOR`, mientras que el *carry* resultante se hace con la compuerta `AND`[1].

[1] **¿Por qué?** Porque si ambos bits de entrada son iguales a 1, entonces sabemos que la suma resultará en 0, "sobrando" una unidad (como lo vemos en la suma tradicional).

Sean $A$ y $B$ dos números de dos bits de la forma $A_1A_0$ y $B_1B_0$. Si queremos obtener el número $C = A * B$, seguimos el siguiente procedimiento:

- Multiplicamos el bit menos significativo de $B$ por los dos bits de $A$. De esta forma, tendremos $A_0$ `AND` $B_0 = C_0$, y $A_1$ `AND` $B_0 = C_1^{'}$.
- Multiplicamos ahora el bit más significativo de $B$ por los dos bits de $A$. De esta forma, tendremos $A_0$ `AND` $B_1 = C_1^{''}$ y $A_1$ `AND` $B_1 = C_2^{'}$
- El bit menos significativo de nuestro resultado será $C_0$. Luego, el bit siguiente se obtiene de la siguiente forma: $C_1 = C_1^{'}$ `XOR` $C_1^{''}$ (tal como lo hacemos con la suma). Como esto nos puede generar un carry, tomamos $C_2^{''} = C_1^{'}$ `AND` $C_1^{''}$
- Ahora, el siguiente bit lo conseguimos como la suma entre el producto de los bits más significativos de cada número, sumado al carry anterior: $C_2 = C_2^{'}$ `XOR` $C_2^{''}$.
- Finalmente, como nuestro número puede tener máximo 4 bits (11 * 11 = 1001), tomamos el bit más significativo del resultado como el carry de la última suma: $C_3 = C_2^{'}$ `AND` $C_2^{''}$.

Finalmente, nuestro circuito queda de la siguiente forma:

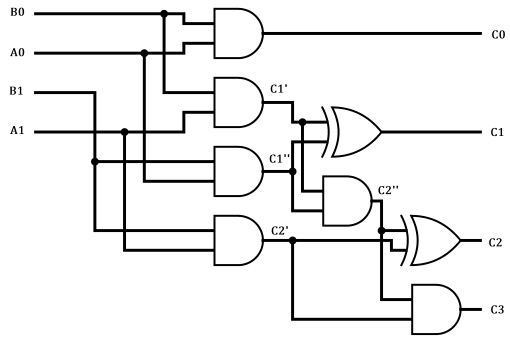


**Figura 1:** Resultado del circuito descrito.

Notar que este diagrama se puede simplificar haciendo uso de *half-adders* y *full-adders*.

b. Describa los valores de salida $Y$ para las siguientes compuertas con sus respectivas señales en los momentos *a, b, c, d, e, f, g, h.*

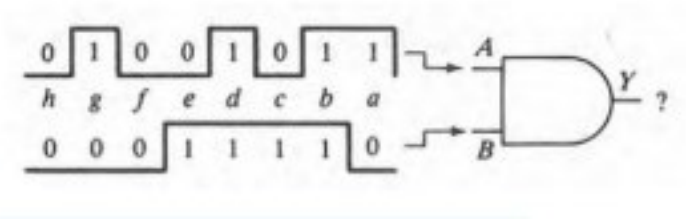


**Figura 2:** Circuito AND

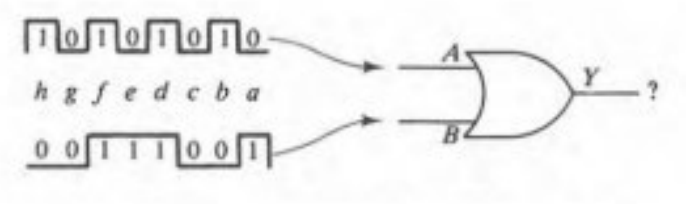


**Figura 3:** Circuito OR

Debido a que las compuertas reciben de a dos valores a la vez, basta con aplicar la propiedad de la compuerta para las señales de cada momento para obtener el valor de salida de ese momento.
Para el circuito AND de la Figura 1, los valores para cada momento son los siguientes:
(a): 0, (b): 1, (c): 0, (d): 1, (e): 0, (f): 0, (g): 0, (h): 0
Para el circuito OR de la FIgura 2, los valores para cada momento son los siguientes:
(a): 1, (b): 1, (c): 0, (d): 1, (e): 1, (f): 1, (g): 0, (h): 1

4. **Ejercicios propuestos**

a. Escriba la representación del número decimal 8 en base 8, la representación del número decimal 14 en base 14 y la representación del número decimal 3 en base 3.
¿Nota algún patrón entre las representaciones?

Las representaciones de los tres números en sus respectivas bases son 10! Ésto pasa porque para cualquier número $\beta$, siempre se dará que

$$1 \times \beta^1 + 0 \times \beta^0 = \beta$$

Por lo que la representación de cualquier número en su propia base es 10.

b. **(T1 - II/2018)** ¿Para qué números $\alpha \in R$ existe un $\beta$ tal que $\alpha = 10_\beta$? Indique una expresión analítica que caracterice a $\beta$ en función de $\alpha$.

Sabemos que

$$\alpha = 10_\beta$$

Si lo pasamos a decimal:

$$\alpha = 1 \times \beta^1 + 0 \times \beta^0$$
$$\alpha = \beta$$

Por lo tanto, para todo $\alpha = \beta$, $\alpha = 10_\beta$

c. Diseñe un Full Subtractor de 4 bits.

Debido a la forma en que construímos la resta entre dos números, para hacer un 4-bit subtractor solamente debemos alterar el 4-bit adder para que sume el primer número de 4 bits con el complemento a 2 del segundo número de 4 bits. Y para conseguir el complemento a 2 del segundo número, nos basta con negar todos sus bits y agregar un carry a la suma! (ya que el complemento a 2 es la negación de todos los bits + 1). Por lo mismo, podemos usar un 4-bit adder y algunas compuertas lógicas para lograr lo que estamos intentando hacer. Para hacer un 4-bit adder, debemos crear primero un Half-Adder, con lo que podemos armar un Full-Adder y con 4 de ellos podemos armar nuestro 4-bit adder. Finalmente, negando las entradas del segundo número y agregando un carry a la suma, obtenemos nuestro 4-bit subtractor

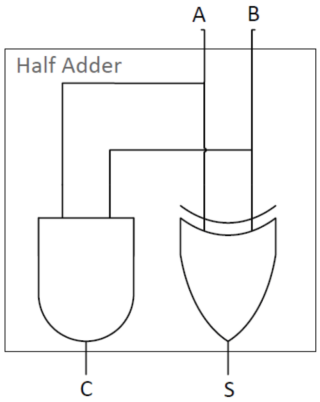


**Figura 4:** Half-Adder

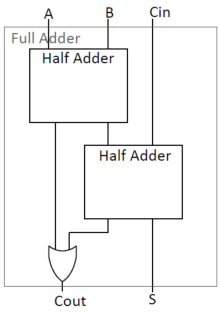


**Figura 5:** Full-Adder

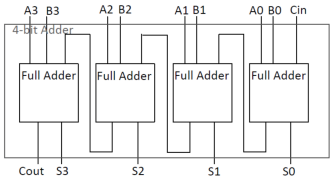


**Figura 6:** 4-bit Adder

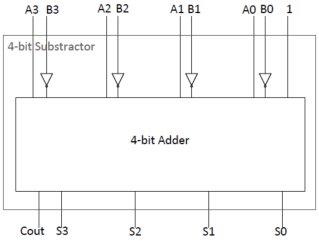


**Figura 7:** 4-bit Subtractor

d. **(T1 - II/2018)** Al utilizar complemento a 2 para representaciones posicionales binarias, se genera una representación no equilibrada, donde existen más elementos negativos que positivos. Modifique el algoritmo de transformación, tal que ahora existan más números positivos que negativos.

Basta con realizar un simple cambio al algoritmo de complemento a 2 visto en clases:

- Anteponer un 1 al número (y no un 0, como en el algoritmo original).
- Negar todos los bits.
- Sumar 1.

Por construcción, se puede ver entonces que se sigue respetando el resultado nulo ante la suma entre un número y su inverso aditivo obtenido a partir de este algoritmo.